Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

ПРОЕКТНА ЗАДАЧА

по предметот

НЕСТРУКТУРИРАНИ БАЗИ НА ПОДАТОЦИ

на тема

ПЕРФОРМАНСНА АНАЛИЗА НА KEY-VALUE БАЗИ НА ПОДАТОЦИ: СПОРЕДБА НА REDIS И POSTGRESQL

Изработка:

Ангела Иванова 211104

Бојан Ристов 211151

Данче Јованова 211156

Моника Габрешанец 211091

Ментор:

Проф. д-р Слободан Калајџиски

Скопје, септември 2025

Содржина

1. Вовед 3
2. Методологија 4
3. Инсталација и конфигурација на PostgreSQL 5
   3.1. Моделирање на податоците 6
   3.2. Импортирање на податоците 6
4. Инсталација и конфигурација на Redis 7
   4.1. Моделирање на податоците 8
   4.2. Импортирање на податоците 8
5. Прашалници 9
   5.1. Едноставни прашалници 9
   5.2. Сложени прашалници 9
   5.3. Многу сложени прашалници 9
6. Перформансна анализа 10
   6.1. Клучни резултати по категории 10
   6.2. Визуелизација и трендови 10
7. Заклучок 12
8. Користена литература: 13

1. Вовед

Во ерата на големи податоци (Big Data), количината, разновидноста и брзината на генерирање информации постојано растат, што создава предизвици за традиционалните релациони бази во однос на скалабилноста, флексибилноста и перформансите. За надминување на овие ограничувања, се развија алтернативни модели како NoSQL базите.

Меѓу нив, Key-Value базите се едни од најприменуваните, со едноставна архитектура заснована на парови клуч–вредност. Тие нудат висока брзина, лесна употреба и хоризонтална скалабилност, што ги прави погодни за апликации со високи перформансни барања. Примери се Redis, Amazon DynamoDB и Riak, кои се користат за кеширање, сесиски менаџмент и обработка на големи податоци.

Паралелно, PostgreSQL, иако релациона база, еволуираше во хибридна платформа со поддршка за NoSQL функционалности. Особено значаен е JSONB типот, кој овозможува ефикасно складирање, индексирање и пребарување на полуструктурирани податоци, па така PostgreSQL може да се користи и како Key-Value решение.

Овој проект има за цел компаративна анализа на перформансите на Redis – како native Key-Value база – и PostgreSQL – како релациона со Key-Value поддршка. Анализата ќе се изврши врз реален датасет со филмски метаподатоци од IMDB и TMDB со над еден милион записи и повеќе од четириесет атрибути, со цел да се утврдат предностите, ограничувањата и препораките за нивна практична примена.

2. Методологија

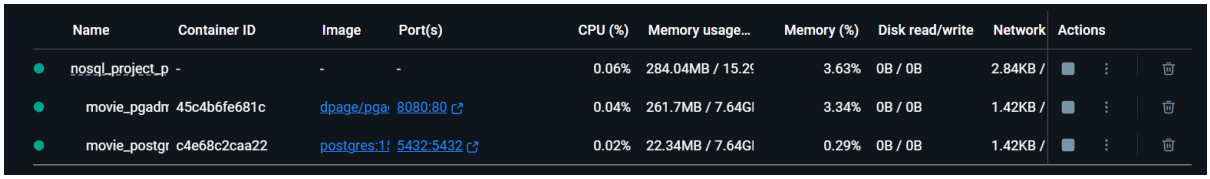
Проектната задача се реализира преку практична имплементација на Key-Value модели во две различни бази на податоци: Redis (како native NoSQL решение) и PostgreSQL (како релациона база со NoSQL проширувања). Методологијата опфаќа неколку чекори:

1. **Инсталација и конфигурација на базите на податоци** – за обезбедување конзистентна околина со репродуцибилни резултати се користеше Docker, со што секоја база беше подигната како изолиран сервис.
2. **Моделирање на податоците** – бидејќи дадениот датасет содржи повеќе од еден милион филмски записи со 42 атрибути, беше потребно внимателно да се дизајнира шемата за складирање, со поддршка и за традиционален релационен модел и за Key-Value претставување.
3. **Импортирање на податоците** – податоците беа прочистени, трансформирани и импортирани во базите со помош на Python скрипти. Во PostgreSQL беа применети и техники за батч-импортирање и автоматско пополнување на Key-Value табела.
4. **Избор и извршување на сценарија за пристап до податоците** – беа дефинирани повеќе сценарија за пристап (едноставни пребарувања, филтрирања, агрегирани извештаи), кои се извршуваат над двете бази.
5. **Перформансна анализа** – мерење и споредба на времето на извршување и ресурсната потрошувачка на двете решенија.

Со ваквиот пристап, овозможена е систематска анализа и изведување релевантни заклучоци за соодветноста на Redis и PostgreSQL во контекст на Key-Value операции врз големи податочни множества.

3. Инсталација и конфигурација на PostgreSQL

PostgreSQL беше инсталиран преку **Docker Compose**, со што се овозможува брзо подигање, изолација и лесна репродукција на околината.



Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memory (%)	Disk read/write	Network	Actions
nosql_project_p	-	-	-	0.06%	284.04MB / 15.2	3.63%	0B / 0B	2.84KB /	
movie_pgadm	45c4b6fe681c	dpage/pga	8080:80	0.04%	261.7MB / 7.64GI	3.34%	0B / 0B	1.42KB /	
movie_postgr	c4e68c2caa22	postgres:1	5432:5432	0.02%	22.34MB / 7.64GI	0.29%	0B / 0B	1.42KB /	

```yaml
services:
  postgres:
    image: postgres:15-alpine
    container_name: movie_postgres
    environment:
      POSTGRES_DB: movie_db
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
      POSTGRES_HOST_AUTH_METHOD: trust
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./scripts:/scripts
    restart: unless-stopped

  pgadmin:
    image: dpage/pgadmin4:latest
    container_name: movie_pgadmin
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@admin.com
      PGADMIN_DEFAULT_PASSWORD: admin123
    ports:
      - "8080:80"
    depends_on:
      - postgres
    restart: unless-stopped

volumes:
  postgres_data:
```

Конфигурацијата е дефинирана во docker-compose.yaml, каде што:

- Се користи официјалниот image postgres:15-alpine, познат по својата стабилност и оптимизираност.

- Дефинирани се основни параметри: име на база (movie_db), корисник (user), лозинка (password).
- Се мапираат два volume директориуми: еден за перзистентно складирање на податоците, и еден за складирање на скриптите.
- Дополнително е вклучен **pgAdmin4** за административна визуелизација и управување со базата преку веб-интерфејс.

Ваквата конфигурација овозможува целосна автоматизација на подигнувањето и унифицирана развојна околина.

3.1. Моделирање на податоците

За PostgreSQL креиравме една табела

1. **movies табела** – класична релациона шема која содржи детален опис на филмовите (ID, наслов, жанр, буџет, приходи, времетраење, продукциски тим итн.).

3.2. Импортирање на податоците

Импортирањето се реализира преку Python скриптата 02_import_data.py, која извршува неколку клучни задачи:

- **Прочистување на податоците**: заменување празни вредности, стандардизација на датуми (release_date), конверзија на булеан полиња (adult, video), ограничување на должината на текстуалните полиња.
- **Мапирање на колони**: бидејќи датасетот содржи повеќе од 40 колони, беше извршено прецизно мапирање кон соодветните полиња во табелата movies.
- **Батч-импортирање**: податоците се внесуваат во групи од по 1000 записи за оптимизација на перформансите. Се користи клаузулата ON CONFLICT (movie_id) DO NOTHING за да се избегнат дупликати.

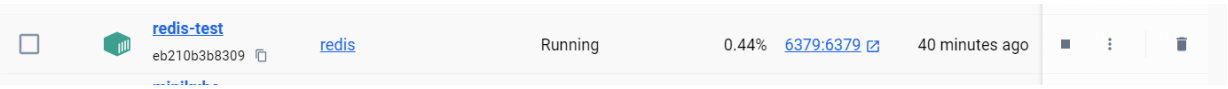
4. Инсталација и конфигурација на Redis

За потребите на проектот, Redis е инсталиран и конфигуриран локално користејќи **Docker Desktop**, што овозможува брза и изолирана средина за работа со базата на податоци.

Redis инстанцата е подигната со командата:

```
docker run --name redis -p 6379:6379 -d redis
```

Оваа команда стартува Docker контејнер со име redis, ја мапира ја стандардната Redis порта од контејнерот на истата порта на локалниот хост и го стартува контејнерот во detached mode.



Поврзувањето со Redis се реализира преку Python библиотеката **redis-py**, користејќи ги стандардните параметри localhost и порт 6379:

За демонстрација, првите 5000 записи од CSV датотека со филмски податоци (IMDB TMDB Movie Metadata Big Dataset) се вчитуваат во Redis. Секој филм се складира како JSON стринг.

4.1. Моделирање на податоците

Податоците се складираат во Redis со **key-value** пристап, каде секој филм има клуч во формат movie:<id>. Дополнително, се користат сетови и сортирани збирки за организирање и индексирање според жанр, актер, година, режисер, приход, профит и други категории.

4.2. Импортирање на податоците

Импортирањето во Redis се реализира со Python скрипта која ги вчитува филмските записи од CSV датотеката и ги зачувува како JSON стрингови со формат на клуч movie:<id>. Податоците автоматски се распределуваат и индексираат по различни категории за полесно пребарување и анализа.

5. Прашалници

За тестирање на функционалностите и перформансите на базите на податоци PostgreSQL и Redis, беа дефинирани и извршени вкупно **9 прашалници**, поделени во три категории според сложеноста на операциите:

5.1. Едноставни прашалници

Овие сценарија се фокусираат на директно извлекување на податоци врз основа на основни филтри:

- **Највисоко оценети филмови (Top-rated movies)** – филмови со рејтинг поголем од 8.0.
- **Филмови со голем буџет (Movies with large budgets)** – филмови со буџет над 100 милиони.
- **Филмови по држава на продукција (Query movies by Production Country)**.

5.2. Сложени прашалници

Комбинација од повеќе услови и агрегирани операции:

- **Анализа на runtime категории (Movies with runtime categories and profitability)** – филмови категоризирани по должина (долги, кратки, стандардни) и нивната профитабилност.
- **Movies by Nolan after 2010** – филмови на режисерот Кристофер Нолан по 2010 година.
- **Финансиски пресметки (Count movies per Certificate)** – број на филмови по категоризација според сертификат.

5.3. Многу сложени прашалници

Напредни аналитички извештаи со агрегирани податоци, групирање и статистички мерења:

- **Детален годишен извештај** – број на филмови, просечен рејтинг и финансиски показатели по година.
- **Top 5 most common keywords** – најчесто користени клучни зборови во филмовите.

- **Average rating per year** – просечен рејтинг по година.

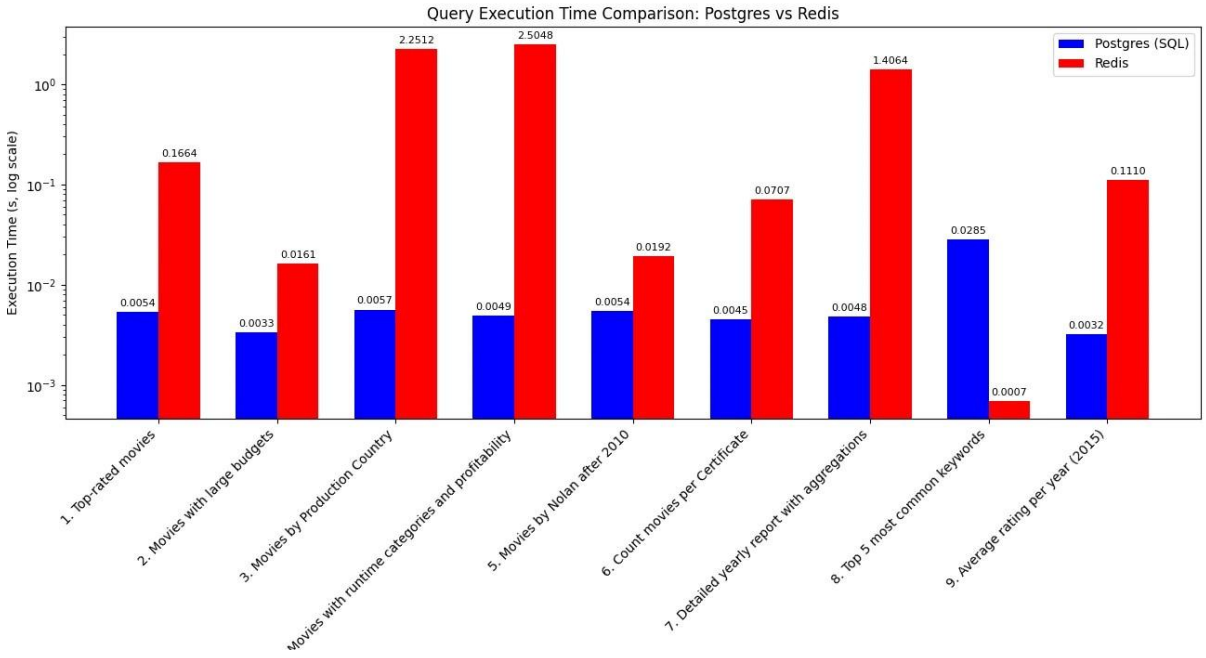
Оваа класификација овозможува систематска анализа на перформансите на различни типови прашалници и обезбедува рамнотежа помеѓу едноставни и комплексни операции.

6. Перформансна анализа

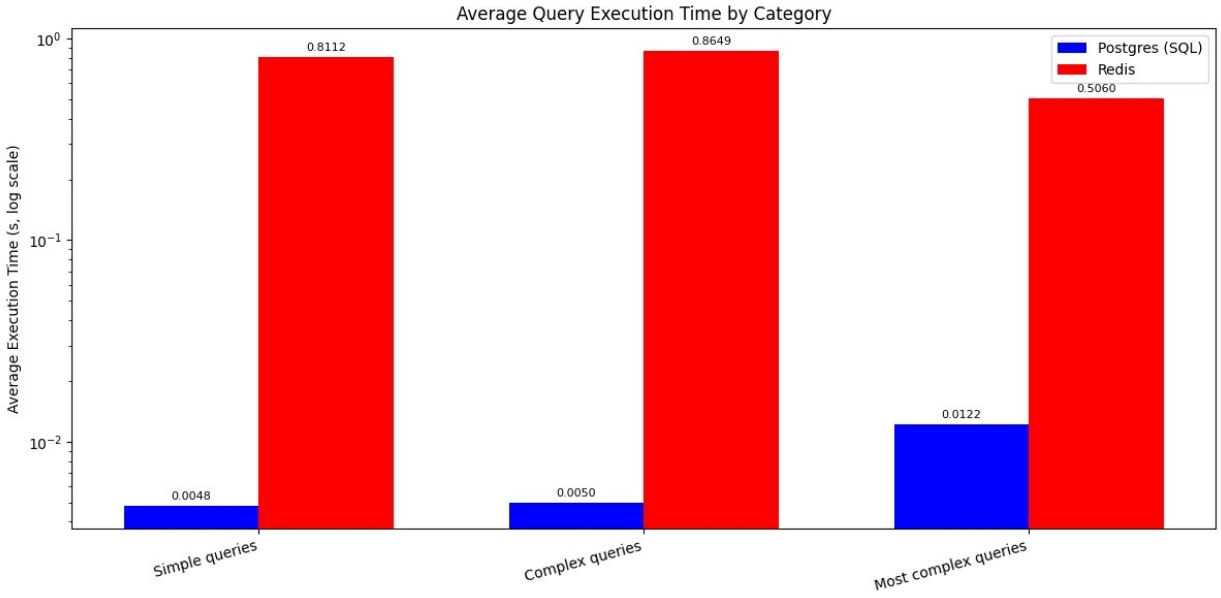
6.1. Клучни резултати по категории

- **Едноставни прашалници**: PostgreSQL има просечно време на извршување од околу 0.005s, додека Redis во истите случаи е значително побавен со просек од околу 0.81s.
- **Сложени прашалници:** PostgreSQL повторно покажува стабилни резултати со просек од околу 0.005s, додека кај Redis просекот се движи околу 0.86s.
- **Многу сложени прашалници**: Кај PostgreSQL времето расте, но останува релативно ниско со просек од 0.012s, додека Redis има просечно време од околу 0.51s.

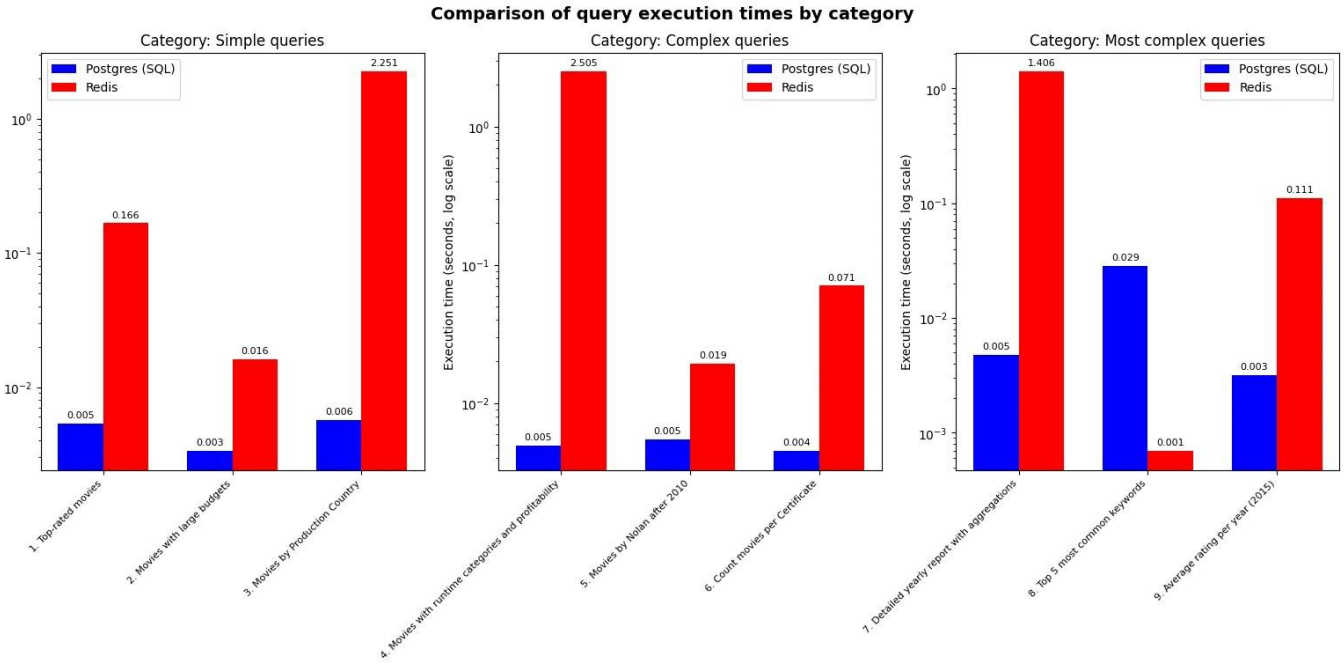
6.2. Визуелизација и трендови



Слика 1 - Споредба на време на извршување на прашалници



Слика 2 - Споредба на време на извршување на прашалници по категорија



Слика 3 - Споредба на време на извршување на прашалници по категорија

7. Заклучок

Преку оваа проектна задача беше спроведена детална анализа и споредба на перформансите помеѓу релационата база на податоци **PostgreSQL** и key-value базата **Redis**. Главната цел беше да се моделираат, складираат и извршуваат различни типови на прашалници врз филмски метаподатоци, со посебен акцент на ефикасноста на извршувањето.

Резултатите од мерењата покажаа дека **PostgreSQL е значително побрз** при сите категории на прашалници – едноставни, сложени и многу сложени. Времињата на извршување се движат во опсег од **0.005 до 0.012 секунди**, што укажува на висока оптимизација на релациониот модел за аналитички и агрегатни операции.

Од друга страна, **Redis**, иако е познат по својата in-memory архитектура и ефикасност кај едноставни key-value операции, покажа значително подолги времиња на извршување за комплексни прашалници. Ова укажува дека Redis е попогоден за кеширање и брз пристап до поединечни податоци, додека PostgreSQL е супериорен за аналитички и извештајни задачи.

8. Користена литература:

1. IMDB & TMDB Movie Metadata Big Dataset (https://www.kaggle.com/datasets/shu bhamchandra235/imdb-and-tmdb-movie-metadata-big-dataset-1m)
2. Redmond, E., & Wilson, J. R. (2012). *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*. Pragmatic Bookshelf.
3. Sadalage, P. J., & Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional.
4. Kamp, M. (2011). *Redis in Action*. Manning Publications.
5. Momjian, B. (2001). *PostgreSQL: Introduction and Concepts*. Addison-Wesley.
6. Chen, R., & Chen, J. (2016). "Performance comparison of NoSQL databases." *International Conference on Computer Science and Technology*.
7. Redis Official Documentation. (2024). Retrieved from (https://redis.io/documentation)
8. PostgreSQL JSON Types Documentation. (2024). Retrieved from (https://www.postgresql.org/docs/current/datatype-json.html)